

PATENT ABSTRACTS OF JAPAN

(11)Publication number : **08-339296**

(43)Date of publication of application : **24.12.1996**

(51)Int.Cl.

G06F 9/06

G06F 9/45

(21)Application number : **08-016858**

(71)Applicant : **INTERNATL BUSINESS MACH
CORP <IBM>**

(22)Date of filing : **01.02.1996**

(72)Inventor : **DUNCAN ROBERT P
FULTON MIKE S
MINCHAU BRIAN J**

(30)Priority

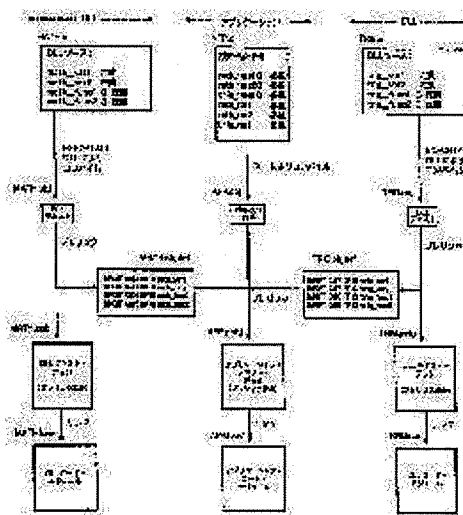
Priority number : **95 2143488** Priority date : **27.02.1995** Priority country : **CA**

(54) METHOD FOR LINKING DYNAMIC LINK LIBRARY TO PROGRAM

(57)Abstract:

PROBLEM TO BE SOLVED: To link symbols within a dynamic link library without necessitating the support of a system linker or a loader.

SOLUTION: This method is provided with a step for generating a definition file including newest information concerning the set of usable external symbols within a library by a prelinker corresponding to each dynamic link library (DLL), and a step for compiling an application program code to a compile code including a code calling a trigger routine. In addition the method is provided with a step a prelinking step for prelinking the application program code to a definition file to generate an object deck including an export definition section, a step for linking the application program code by means of the system linker and an execution and solution step for executing the code, solving the reference of an unsolved symbol within the code by a trigger routine, and providing a symbol to an application program without necessitating the support of the system linker or loader.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

特開平8-339296

(43) 公開日 平成8年(1996)12月24日

(51) Int.Cl. ⁸	識別記号	序内整理番号	F I	技術表示箇所
G 0 6 F 9/06 9/45	4 1 0	9189-5B	G 0 6 F 9/06 9/44	4 1 0 E 3 2 2 A

審査請求 未請求 請求項の数5 O L (全 15 頁)

(21) 出願番号 特願平8-16858

(22) 出願日 平成8年(1996)2月1日

(31) 優先権主張番号 2 1 4 3 4 8 8

(32) 優先日 1995年2月27日

(33) 優先権主張国 カナダ (CA)

(71) 出願人 390009531

インターナショナル・ビジネス・マシーンズ・コーポレーション

INTERNATIONAL BUSINESS MACHINES CORPORATION

アメリカ合衆国10504、ニューヨーク州
アーモンク (番地なし)

(72) 発明者 ロバート・ポール・ダンカン

カナダ、オンタリオ州スカボロ、ジェネラ・スクエア 127

(74) 代理人 弁理士 合田 潔 (外2名)

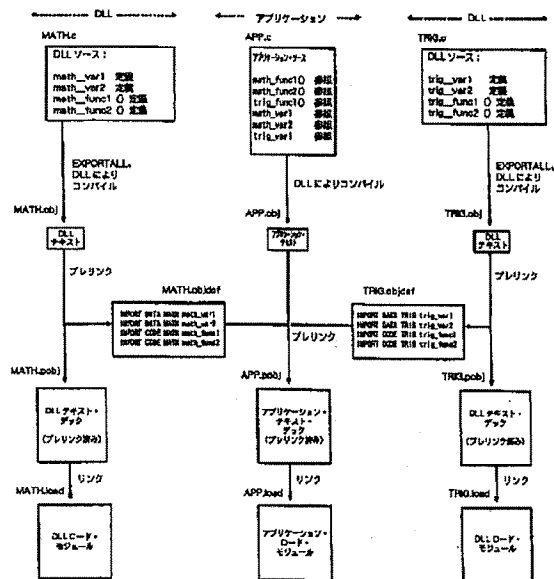
最終頁に続く

(54) 【発明の名称】 動的リンク・ライブラリをプログラムにリンクする方法

(57) 【要約】

【課題】 システム・リンカまたはローダのサポートを必要とせずに、動的リンク・ライブラリ内の記号をリンクする。

【解決手段】 本方法は、各動的リンク・ライブラリに対応して、プレリンカによりライブラリ内で使用可能な外部記号のセットに関する最新の情報を含む定義ファイルを生成するステップと、コンパイラによりアプリケーション・プログラム・コードを、トリガ・ルーチンと呼出すコードを含むコンパイル・コードにコンパイルするステップと、アプリケーション・プログラム・コードを定義ファイルにプレリンクし、エクスポート定義セクションを含むオブジェクト・デックを生成するプレリンク・ステップと、システム・リンカによりアプリケーション・プログラム・コードをリンクするステップと、前記コードを実行し、トリガ・ルーチンにより前記コード内の未解決記号の参照を解決し、システム・リンカまたはローダの支援を必要とせずに、アプリケーション・プログラムに前記記号を提供する、前記実行及び解決ステップを含む。



【特許請求の範囲】

【請求項1】動的にリンクされるライブラリをデータ処理オペレーティング・システム上で実行されるプログラムにリンクする改良された方法であって、前記データ処理オペレーティング・システムがコンパイラ、プレリンカ、実行時ライブラリ、システム・リンカ及びシステム・ローダを有するものにおいて、各動的リンク・ライブラリに対応して、前記プレリンカにより、前記ライブラリ内で使用可能な外部記号のセットに関する情報を含む定義ファイルを生成するステップであって、前記プレリンカが前記定義ファイルを前記動的リンク・ライブラリ内の任意の変更を反映して最新に維持する、前記生成ステップと、前記コンパイラによりアプリケーション・プログラム・コードをコンパイルするステップであって、前記コンパイル・コードがトリガ・ルーチンを呼び出すコードを含む、前記コンパイル・ステップと、前記アプリケーション・プログラム・コードを前記定義ファイルにプレリンクするステップであって、エクスポート定義セクションを含み、前記システム・リンカによりリンクされるオブジェクト・デックを生成する、前記プレリンク・ステップと、前記システム・リンカにより、前記アプリケーション・プログラム・コードをリンクするステップと、前記コードを実行し、前記トリガ・ルーチンにより前記コード内の未解決記号の参照を解決するステップであって、前記トリガ・ルーチンが前記プレリンカにより生成された前記エクスポート定義セクションをアクセスすることにより、前記実行時ライブラリが前記未解決記号を含む前記動的リンク・ライブラリをアクセスし、前記動的リンク・ライブラリをロードし、前記記号参照を解決することを可能にし、前記システム・リンカまたはローダの支援を必要とせずに、前記アプリケーション・プログラムに前記記号を提供する、前記実行及び解決ステップと、を含む、方法。

【請求項2】前記外部記号が外部変数及び外部関数を含む、請求項1記載の方法。

【請求項3】前記トリガ・ルーチンが変数の参照に関するトリガ・ロードまたは関数の呼び出しに関するトリガ・ロードである、請求項1記載の方法。

【請求項4】前記定義ファイルが各特定の記号がコードかデータかを指定し、前記記号を定義する前記動的リンク・ライブラリを指定する、請求項1、2または3記載の方法。

【請求項5】前記エクスポート定義セクションが前記定義ファイルからの情報を含む、請求項1、2または3記載の方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は一般に動的リンク・ライブラリに関し、特に、システム・リンカまたはローダのサポートを必要としない動的リンク・ライブラリに関する。

【0002】

【従来の技術】コンピュータ・プログラムを作成するとき、プログラムは高級言語によりソース・コード・ファイルを作成する。しばしば使用されるこれらの高級言語に、“C”及び“C++”がある。これらの言語ではプログラムは通常、特定のアプリケーションまたは関数を実行するためのコードを作成する。このソース・コード・ファイルは次にコンパイラに提供され、コンパイラはソース・コードを入力として受け取り、2進オブジェクト・デックを生成する。これらの2進オブジェクト・デックの多くはマシン実行可能プログラムを完成するように結合される。この完全な実行可能プログラムまたはロード・モジュールを構成するために、リンカが別々のオブジェクト・デックを単一の関数プログラムに結合する。各オブジェクト・デックは名前により記号を定義及び参照する。リンカはまた、異なるオブジェクト・デック内に存在しうる定義及びそうした定義の参照を解決する。

【0003】コンパイラは、変更可能データ領域内の記号を参照するオブジェクト・デック、並びにシステム・リンカ/ローダによりサポートされない形式を生成する。これらの記号参照は後に定義とリンクされたり、部分的にリンクされたままである。部分的にのみリンクされる場合、そのリンクは実行時に解決される。

【0004】現コンパイラでは、オブジェクト・デックがリンカによりリンクされるとき、記号定義を解決することが必要である。例えばIBMから提供されるS/370データ処理システム(DPS)で使用されるコンパイラを参照されたい。このシステムでは、オブジェクト・デックのコピーが、それらを使用する各アプリケーションに静的にリンクされなければならない。

【0005】各オブジェクト・デックを、それらを使用する各アプリケーションに静的にリンクするために、多くの問題が生じる。例えば多くのオブジェクト・デックのコピーを有することが必要となり、オブジェクト・コード・ファイルのサイズが非常に大きくなる。リンク時に定義の1つを必要とする様々なプログラムの各々が、使用可能なオブジェクト・デックのコピーを有さねばならず、それを可能にする唯一の方法はオブジェクト・デックのコピーを作成することによる(その際、オブジェクト・デックがアプリケーションに静的にリンクされねばならない)。明らかに、各アプリケーションがそれぞれコピーを保有する必要無しに、各アプリケーションが使用可能な各オブジェクト・デックの単一のコピーを有することが望ましい。

【0006】複数のコピーを突き止め、修正しなければならない場合には、デック内のコードを保守することが

非常に困難になる。単一のコード片を修正するためには、そのコードを使用する全てのプログラムを静的に再リンクする必要がある。コンパイル単位が1箇所のみ保持されていると、修正は比較的容易になる。

【0007】動的リンク・ライブラリ（DLL）は、ロード・モジュールとして集められる1つ以上の関数または変数の集合である。このように集められたロード・モジュールは、別々のアプリケーション・ロード・モジュールまたは別のDLLからアクセスされうる実行可能コードを含む。アプリケーションからDLL内の関数または変数への接続（リンク）は、アプリケーション生成時に静的に行われるのではなく、アプリケーションの実行の間に動的に行われる。

【0008】動的リンク・ライブラリ（DLL）は、参照記号のリンクをアプリケーション実行時まで遅延させるために使用される。本明細書では、用語“記号（symbol）”は、DLL内に保持される関数及び変数のいずれかまたは両者を定義するために使用される。これは全てのアプリケーションが同一のDLLを使用することを可能にする。このことは、参照記号の定義を含む各オブジェクト・デックの複数のコピーを記憶及び保持する問題を解決する。DLLの使用はまた、別のロード・モジュール内の定義をアクセスするために、あるロード・モジュール内での関数呼び出しまたは変数参照を可能にする。

【0009】大部分のDLLにおけるこうした実現は、実行時におけるロード・モジュール内の記号のアドレス変更を要求する。これはプログラムを再入不可能にし、全てのユーザがDLLの同一のコード領域を共有できなくなる。本発明は、DLLの変更がロード・モジュール内の変更を要求しないことを保証する方法を提供する。このことはプログラムを再入可能にする。

【0010】本発明はプレリンカ（prelinker）を用いてコンパイラにより生成されるオブジェクト・デックを入力として受け取り、プレリンク・オブジェクト・デックを出力として生成する。プレリンク・オブジェクト・デックはそのコード領域に変更可能データ領域に関する情報を含む。コード領域は、システム・リンカ/ローダが受諾可能な標準形式である。コード領域は、コード領域内のアドレス情報が変更されないように書き込み保護されてもよい。プレリンカはデータ領域に関する情報をコード領域内に集め、それにより実行時ライブラリが必要に応じてデータを突き止めることができるようになる。プレリンカにより集められた情報は、参照記号が動的にリンクされるときに、実行時ライブラリが変更可能データ領域の必要サイズ、その初期化方法、及びデータ領域内のアドレスの取り決めを決定できるようにする。

【0011】本発明によれば、システム・リンカ/ローダは、動的にリンクされないコード領域内の記号のアドレスを取り決めるだけでよい。ローダがアプリケーションのコード領域をロードすると、実行時にアドレスが変

更されないようにコード領域が書き込み保護されてもよい。

【0012】実行時ライブラリ（RTL）は、実行時に動的にリンクされる記号のアドレスを、アプリケーションの変更可能データ領域内で取り決める。

【0013】本発明は、ユーザ・ソース・コード内の変更を必要とせずに、記号が動的にリンクされることを可能にする。もちろんアプリケーション作成者が明示的なRTLサービスを使用するように選択し、参照記号のアドレスを獲得してもよい。

【0014】本発明は以下の出願に関連する。L. Hallによる米国特許出願番号第5175828号は、プログラムのデータ及び状態を保存するがプログラム関数（コード領域）内の命令を変更する、動的リンカについて述べている。James G. Littletonによる同第5247678号は、コード領域内でのアドレス取り決めを行う動的リンカについて述べている。結果的にアプリケーションが再入不可能となる。Daniel L. Murphyによる同第5297291号は、記号ベクトルを構成するためにプログラマにより提供される、固定順序のエクスポート記号の順序付けリストについて述べており、リンカ及びオペレーティング・システムがこのベクトルを使用する。Kennith Lundinによる同第5339430号は、アプリケーションにより使用される間に、ソフトウェア・モジュール（ライブラリ）の新たなバージョンを導入するために要求される動的リンクについて述べている。

【0015】

【発明が解決しようとする課題】本発明の主な目的は、コンパイラ、プレリンカ及び実行時ライブラリの使用により、アプリケーション実行時に別のプログラムを動的にロードするためにプログラムを部分的にリンクし、システム・リンカまたはローダのサポートを要求せずに、部分的にリンクされたプログラム内のアドレスを解決する方法を提供することである。これはアプリケーション・プログラムのソース・コードの変更無しに達成される。

【0016】本発明の別の目的は、関数に関する情報が実行可能コードのアドレスだけでなく、関数のコンテキストまたは環境に関する情報も含む間接方法（関数記述子）を提供することである。コンテキスト情報または環境情報が、関数コード領域に対応するデータ領域を指し示し、実行時に関数を解決するためにロードされる必要のあるロード・モジュールを識別する。

【0017】本発明の更に別の目的は、変数に関する情報がそのコンテキストまたは環境に関する情報を含む間接方法（変数記述子）を提供することである。コンテキスト情報または環境情報が、実行時に変数を解決するためにロードされる必要のあるロード・モジュールを示す。

【0018】本発明の更に別の目的は、実行時ライブラ

リにより割当てられるデータ領域内でのアドレス解決を可能にする。プログラムのコード領域はアドレス解決のためには使用されず、コード領域は1度コンピュータ・メモリ内にロードされると、複数呼び出しにより同時並行に共有される。このことはプログラムを再入可能にする。

【0019】本発明の別の特長は、DLLからインポートされる関数を参照するプログラムが、そのDLLをロードする必要がないことである。関数ポインタ（関数記述子のアドレス）が、最初の呼び出しを行い、DLLロードをトリガする別のプログラムに渡される。

【0020】本発明の更に別の目的は、実行の目的のために、関数のアドレスの代わりに関数記述子のアドレスの使用を可能にすることである。

【0021】本発明は更に、DLLが他のDLLから記号をインポートする能力を提供する。本発明は更に、

（関数記述子ベースの）DLLイネーブル化オブジェクト・コードが、（関数アドレス・ベースの）非DLLオブジェクト・コードと最小の制限により一緒に使用されるようにする。関数ポインタの比較のためにではなく、関数の実行目的のために関数記述子が非DLLコードに渡される。非DLLコード内の関数アドレスは、実行のためにも関数ポインタ比較のためにもDLLコードへは渡されない。

【0022】本発明は更に、コンパイル時に使用される記号タイプを知ること無しに、インポート記号または非インポート記号によるDLLオブジェクト・コード内の記号参照の解決を可能にする。

【0023】本発明では更に、DLLサービスのための明示的RTL関数呼び出しを有するコードが、暗黙的DLLオブジェクト・コードを有するコードと混合される。

【0024】

【課題を解決するための手段】本発明は、動的にリンクされるライブラリをデータ処理システム上で実行されるプログラムにリンクする改良された方法を提供し、データ処理システムがコンパイラ、プレリンカ、実行時ライブラリ、システム・リンカ及びシステム・ローダを有する。本方法は次のステップを含む。各動的リンク・ライブラリに対応して、プレリンカにより、ライブラリ内で使用可能な外部記号のセットに関する情報を含む定義ファイルを生成するステップであって、前記プレリンカが前記定義ファイルを前記動的リンク・ライブラリ内の任意の変更を反映して最新に維持する前記生成ステップと、前記コンパイラにより、アプリケーション・プログラム・コードをコンパイルするステップであって、前記コンパイル・コードがトリガ・ルーチンを呼び出すコードを含む前記コンパイル・ステップと、前記アプリケーション・プログラム・コードを前記定義ファイルにプレリンクするステップであって、エクスポート定義セクシ

ョンを含み、前記システム・リンカによりリンクされるオブジェクト・デックを生成する前記プレリンク・ステップと、前記システム・リンカにより前記アプリケーション・プログラム・コードをリンクするステップと、前記コードを実行し、前記トリガ・ルーチンにより前記コード内の未解決記号の参照を解決するステップであって、前記トリガ・ルーチンが前記プレリンカにより生成された前記エクスポート定義セクションをアクセスすることにより、前記実行時ライブラリが前記未解決記号を含む前記動的リンク・ライブラリをアクセスし、前記動的リンク・ライブラリをロードし、前記記号参照を解決することを可能にし、前記システム・リンカまたはローダの支援を必要とせずに前記アプリケーション・プログラムに前記記号を提供する、前記実行及び解決ステップと、を含む。

【0025】

【発明の実施の形態】コンパイラが、変更可能データ領域内の記号（通常はデータ項目）及びコード領域内の記号（通常は実行可能コード）を参照するオブジェクト・コードを参照する。

【0026】S/370 DPS上のオブジェクト・モジュールは、名前を項目のID番号に関連付ける名前レコード、項目の一部の初期化情報に一定値を提供する初期化レコード、並びにある項目のアドレスを別の項目内の特定のロケーションへ配置することを要求する取り決め（fix up）レコードを含む。

【0027】S/370 DPS上の各名前レコードは更に、名前をタイプに関連付ける。幾つかのタイプの例として、コード領域内の項目の定義、コード領域内の項目の参照、変更可能データ領域内の項目の定義、または変更可能データ領域内の項目の参照がある。

【0028】更にS/370 DPSでは、コード領域内で定義される項目の初期化情報がオブジェクト・コード命令を含む。

【0029】S/370 DPS内のシステム・リンカ／ローダは、変更可能データ領域内の項目の初期化をサポートしない。非DLLオブジェクト・コード（旧タイプ）だけが、リンク時に解決されねばならない命名記号を参照できる。

【0030】関数または外部変数は、それらの定義が参照プログラム内に存在しない場合、“インポート”される。関数または外部変数は、特定ロード・モジュール内でのそれらの定義が別のロード・モジュール内で参照される場合、“エクスポート”される。

【0031】記号を別のロード・モジュールから動的にリンクするために、アプリケーション・ソース・ファイルが“DLL”オプションと一緒にコンパイルされねばならない。このオプションはコンパイラに、関数を呼び出すための、及びアプリケーション・モジュールに動的にリンクされるDLLモジュール内に配置される外部変数

を参照するための、DLLオブジェクト・コードを生成するように命令する。

【0032】次のコード・フラグメントは、IBMから提供されるMVSオペレーティング・システムにおい

```
math_func1(. . . .); /*関数プロトタイプ他と差異無し*/
extern int math_var1; /*宣言 - 差異無し*/ main() {
:
math_func1(); /*インポート関数参照*/
:
math_var1=7; /*インポート変数参照*/
:
}
```

【0034】S/370 DLLオブジェクト・デックでは、関数を参照するためにDLLオブジェクト・コード・シーケンスが常にコンパイラにより生成される。このコードにより、インポート関数が非インポート関数と同様に参照される。

【0035】コンパイラは、関数がアプリケーション内の別のコンパイル単位内で定義されているか、或いはDLLから真にインポートされるのかを知らず、両方に対して同一コードを生成する。呼ばれる関数は、関数記述子内に示される関数である。任意のインポート関数に対して、最初に関数記述子が、RTLトリガ関数が呼ばれるべきことを示す。

【0036】プログラム・モジュールを動的にリンクするための3つの方法が存在する。任意のユーザ・コードが実行される以前に接続が形成される場合、これは暗黙的ロード時 (load-time) DLLと呼ばれる。接続がユーザ・コード実行中に形成されるがソース・コード内に見い出されないRTLサービスを通じてトリガされる場合、これは暗黙的ロード・オン・ユース (load-on-us

て、アプリケーションがDLLを使用する方法を示す。アプリケーションは、あたかも関数が静的に結合されるように作成される。

【0033】例えば、

e) DLLと呼ばれる。接続がユーザ・コード実行の間に、ソース・コード内に見い出される明示的RTLサービスを通じて形成される場合、これは明示的実行時 (runtime) DLLと呼ばれる。

【0037】暗黙的ロード・オン・ユースでは、RTLトリガ関数がDLLコード領域及びデータ領域をロードする。RTLは、システム・ロードによりコード領域をロードすることによりこれを実行し、RTL自身は、ロードされたコード領域内で見い出される情報にもとづき対応するデータ領域を割当て、初期化する。1度ロードされると、RTLは呼び出し側のデータ領域内の記述子を見い出し、記述子内のトリガ関数のアドレスを所望の関数のアドレスにより置換する。従って、DLLのこのトリガは1度だけしか発生しない。この後、あたかもアプリケーションがその最初に所望された呼び出しを行うかのように、RTLがDLL関数を呼び出す。

【0038】S/370 DPSにおいて、関数記述子を通じて関数を呼び出すために生成されるオブジェクト・コードは、次のようである。

```
L Rx, QCON(Rc) /*関数記述子のオフセットを獲得*/
A Rx, DATAADDR(r13) /*関数記述子のアドレスを獲得するために、現ロード*/
/*モジュールのデータ領域の基底アドレスを加算*/
LM r15, r0, 8(Rx) /*関数のアドレス及びそのロード・モジュールの対応*/
/*するデータ領域アドレスを、関数記述子内のオフセ*/
/*ット8及び12のワードからロード*/
BALR r14, r15 /*関数を呼び出す (おそらくRTLトリガ関数) */
```

【0039】データ領域において、対応する関数記述子は次の記載項目を含む。

```
on entry
+0 LR r0, r15 /*この記述子アドレスをr0に保管
+2 L r15, 16(r15) /*@GETFNのアドレスを下記からロード
+6 BR r15 /*@GETFNを呼び出す
+8 Addr(function) /*関数のアドレス
+12 Addr(Data area) /*対応データ領域のアドレス
+16 V(@GETFN) /*RTLルーチン
```

【0040】関数記述子はその最初に実行可能命令を含む。これは実行目的の関数のアドレスの代わりに、関数記述子のアドレスを旧非DLLオブジェクト・コードに使用させるためである。関数記述子の最初の命令を実行

すると、記述子のアドレスがr0に保管され、RTLサービス@GETFNが呼ばれる。@GETFNは、そのアドレスが記述子内においてオフセット8の関数を呼び出す。

【0041】図2は、アプリケーションAPPのデータ

領域内の記述子の使用による、間接アドレス指定を示す。アプリケーションAPPのコードは、その自身のデータ領域内の記述子を参照する。

【0042】図3に示されるように、本発明は旧非DLLオブジェクト・コードと互換である。非DLLオブジェクト・コード内のコード領域内の関数のアドレスは、実行または比較のためにDLLオブジェクト・コードに渡されなくてよい。なぜなら、本発明によればDLLコードはそれを記述子として扱うからである。記述子のアドレスは、実行目的のためにのみ、関数アドレスとして非DLLコードに渡されうる。本発明はこの実行目的のために、各関数記述子の最初に“グルー・コード (glue code)”を含む。

【0043】図3は、関数“sortthem2”内の非DLLコードを示し、これは“comp”と命名される関数のアドレスを非DLLルーチン“qsort”に渡す。関数“sortthem1”内のDLLコードが同一のルーチン“qsort”を呼び出す。これは“comp”に対応する関数記述子のアドレスを渡す。“qsort”内の非DLLコードが記述子内のデータを実行可能コードとして解釈し、それを実行する。関数記述子の最初の“グルー・コード”がこれを成功裡に発生させる。グルー・コードは小さく高速である。

【0044】S/370 DLLオブジェクト・デックでは、現コンパイル単位内で定義されない変数を参照するためにDLLオブジェクト・コード・シーケンスが常

にコンパイラにより生成され、こうした変数がインポートされうる。このコードにより、インポート変数が非インポート変数と同様に参照される。

【0045】コンパイラは、この変数がDLLからインポートされうることを示す参照を発行する。インポート変数の参照は追加の間接方法を通じて解決される。アプリケーションによるDLL変数の参照は、変数記述子内のアプリケーションのデータ領域に記憶されるアドレスを介する間接参照として生成される。参照の生成コード・シーケンスはインポート変数の最初の参照に対するDLLローダの実行を含む。

【0046】例えば、次のようなコード・フラグメントが提供される。

```
.  
.   
.   
extern char * var1;  
.   
.   
.   
var1 = 7;
```

【0047】インポートされうる変数に対して、コンパイラはあたかも以下のようにコード化されたかのように追加レベルの間接方法によりコードを生成する。

```
static char * base_mydata; /*自身のデータ領域の基底アドレス*/  
static char * off_var1; /*基底から変数1データへのオフセット*/  
static char * addr_var1; /*変数1データのアドレス*/  
static char * off_fixup; /*プレリリンクにより取り決められるオフセット*/  
static char * addr_var1_descriptor; /*アドレス変数記述子*/  
if(off_fixup > 0){ /*取り決めの高位ビットがオンでないならば自*/  
off_var1 = off_fixup; /*身のデータ内の変数1へのオフセットを使用*/  
}  
else{ /*取り決めの高位ビットがオンならば、オフセ*/  
addr_var1_descriptor = /*ットは自身のデータ領域内の記述子へのオフ*/  
base_mydata + /*セットであるが、その高位ビットを無視*/  
(0xEFFFFFFF & off_fixup);  
off_var1 = /*記述子内の値を獲得*/  
*addr_var1_descriptor;  
if(off_var1 == 0){ /*DLLがまだロードされていなければ、そ*/  
@TRGLOR(addr_var1_descriptor); /*れをロード*/  
off_var1 = /*記述子内の値を取り決めるために、それを*/  
*addr_var1_descriptor; /*獲得*/  
}  
}  
/*インポートされるか否かにかかわらず、off_var1を割当て*/  
addr_var1 = base_mydata + off_var1;
```



```
*addr_var1 = 7;
```

【0048】暗黙的ロード時DLLまたは暗黙的ロード・オン・ユースDLLにおいて、コンパイラは、取り決められたオフセットがアプリケーション内で定義される変数に対するものか、またはDLLからインポートされる変数記述子に対するものかを実行時にテストするコードを生成する。例えば、取り決められたオフセットの高位ビットがテストされる。オフセットが変数記述子に対するオフセットの場合、コンパイラにより生成されるコードは、変数定義を有するDLLがロードされて変数アドレスが既に解決されたか否かをテストする。このテストは（アプリケーション自身のデータ領域内の）変数記述子を調査することにより実行される。まだ解決されていないければ、変数記述子のアドレスによりRTLサービスを呼び出す。

```
LOADQCON  ICM  r15, QCON(Rc) /*プレリンカにより取り決められるオフセ*/
                                     /*ットをロード*/
                                     BM  DLLREF /*高位ビットがオフならば、通過*/
ADDBASE   LR   Rx, r15 /*オフセットを所望のレジスタにコピー*/
...                                     /*データ領域の基底アドレスを加算*/
...                                     /*アドレスを使用*/
DLLREF     MVC  BSAVE(2, r13), INST_OFF(Rc) /*復帰アドレスを保管*/
           B    DLLREFS /*分岐命令x'477XXXX'をコピー*/
DLLREFS    LA   r15, 0(r15) /*オフセットの高位ビットをクリア*/
           A    r15, DATAADDR(r13) /*データ領域の基底アドレスを加算*/
           ST   r15, REGSAVE+4(r13) /*記述子アドレスを保管*/
LOADADDR   ICM  r15, 0(r15) /*記述子から変数アドレスを獲得*/
           EX   r0, BSAVE(r13) /*非ゼロ（すなわち既に解決済み）ならば*/
                                     /*ADDBASEに分岐*/
                                     /*変数アドレスを解決するためにDLLをロード*/
DLLLOAD    L    r15, V(@TRGLOR) /*DLLロード関数のアドレスをロード*/
           ST   r14, REGSAVE(r13) /*DLLロード以前に、r14を保管*/
           ST   r0, REGSAVE+8(r13) /*DLLロード以前に、r0を保管*/
           L    r0, REGSAVE+4(r13) /*記述子アドレスをロード*/
           BALR r14, r15 /*DLLロード関数を呼び出す*/
           DC   =F'PARMSIZE' /*パラメータ・サイズを8バイトに割当て*/
           LM   r14, r0, REGSAVE(r13) /*レジスタを復元*/
           B    LOADADDR /*LOADADDRに分岐*/
/*命令の増加を低減するために、DLLREFS、LOADADDR、DLLLOADは複数DLL*/
/*変数参照により共有されるが、各変数参照に対して1つのDLLREFセクション*/
/*が存在する*/
```

【0053】DLLの要件の1つは、DLL内に含まれる記号が他のモジュールにより要求されるときにそれらがエクスポートされなければならないことである。コンパイラは、記号がオブジェクト・モジュール内の対応する定義名レコード内にエクスポートされる事実を符号化する。MVSでは、これはEXPORTALLコンパイル・オプションを指定することにより達成される。このオプションはコンパイラに対して、DLL内の全ての外部定義関数及び変数をそのDLLを使用するモジュールにエクス

【0049】RTLサービスは、インポート変数の定義を含むDLLをロードする。RTLサービスは、次に変数のアドレスを変数記述子に記憶する。変数記述子は最小限、RTLが変数のアドレスを配置するロケーションを含む。

【0050】次にオブジェクト・コードが変数記述子からアドレスを獲得し、進行する。

【0051】変数の取り決められたオフセットがインポートされない場合には、コンパイラは自身のデータ領域内に存在する変数のアドレスを獲得する。これら全てはソース・コード・レベルでは見ることができない。

【0052】S/370において変数記述子を使用するために生成されるオブジェクト・コードは、次のようである。

ポートするように命令する。EXPORTALLコンパイル・オプションを用いることにより、DLLを呼び出す最初のモジュールだけではなくDLL内の全ての外部変数及び関数が、そのDLLを使用する全てのモジュールにとって使用可能になる。

【0054】代わりにMVSでは、指定外部変数及び関数だけをDLLからエクスポートするために、#pragma宣言が使用される。これを次のソース・コード例により示す。

```
#pragma export(math_func1)
#pragma export(math_func2)
int math_func1(){
:
:
}
int math_func2(){
:
:
}
int kpItHdn(){
:
:
}
:
:
#pragma export(math_var1)
#pragma export(math_var2)
int math_var1;
int math_var2;
int kpItHdnVar;
```

【0055】上述例では、関数“math_func1()”及び“math_func2()”、並びに変数“math_var1”及び“math_var2”がエクスポートされ、それによりこれらがDLLを使用する全てのモジュールにとって使用可能になる。関数“kpItHdn()”並びに変数“kpItHdnVar”はエクスポートされず、ユーザ・モジュールにとって直接的には使用可能ではない。

【0056】プレリンカ入力はコンパイラにより生成される多くのオブジェクト・デッキを含む。プレリンカ出力はプレリンクされたオブジェクト・デッキを含む。

【0057】プレリンカは、データ領域内のデータ項目をマップする。これはデータ項目にその領域内のオフセットを割当て、出力プレリンク・オブジェクト・モジュール内で、それらをオフセットにより参照するように取り決めることにより、達成される。

【0058】プレリンカはまた、RTLにより使用されるデータ領域内の項目に関し、コンパイラにより生成された初期化情報を収集する。

【0059】コード領域内の記号に対する全てのアドレス取り決めが、単にシステム・リンカ／ローダに管理のために渡される。

【0060】プレリンカは、データ領域内の全ての変数及び関数の初期化を定義する情報を生成する。変数のアドレスがまだ知れていないことを示すように、変数記述子が初期化される。インポート関数の関数記述子は、DLLのロードをトリガするRTLルーチンのアドレスを含むように初期化され、非インポート関数の関数記述子は、同一ロード・モジュール内の関数のアドレスを含むように初期化される。

【0061】変数がインポートされない場合、プレリンカはデータ領域内の変数オフセットによりその参照を取り決める。変数がインポートされる場合には、プレリンカはその参照を、プログラムのデータ領域内のその記述子のオフセットにより取り決め、これが記述子であることを示す（高位ビットがオン）。

【0062】プレリンクの間、全てのエクスポート関数のリスト及び全てのエクスポート変数のリストが収集される。これらのリストを定義するオブジェクト・コードがプレリンカにより生成され、出力プレリンク・オブジェクト・デッキ内にエクスポート定義セクションとして発行される。これらのリストは最終的に、プログラムに関連付けられるロード・モジュールのコード領域内に配置される。RTLはコード領域内のこれらのリストを読み出し、このプログラムによりエクスポートされる関数及び変数の名前、並びにそれらのアドレスを獲得する。

【0063】記号をどこからインポートするかに関する情報が、プレリンク時にプレリンカに提供されなければならない。しかしながらフォーマットはサイドファイル内にある必要はない。図1に示される例では、これはサイドファイルMATH.objdef及びTRIG.objdefを用いて達成される。図示の例では、DLL定義サイドファイル（宣言ファイル）がDLLプロバイダにより提供され、アプリケーションのプレリンク時に使用される。サイドファイルは、アプリケーションによりインポートされうるDLL内の全ての関数及び変数を記述するIMPORT宣言を含む。

【0064】ソース・コード・ファイルAPP.cは他のファイル同様にコンパイルされる。しかしながら、オブジェクト・デッキApp.objがプレリンクされる時、サイドファイルMATH.objdef及びTRIG.objdefは、インポート記号の獲得場所に関する追加の情報を提供する。

【0065】このDLL定義サイドファイルは、DLL内で定義される関数及び変数の参照を解決する方法をプレリンカに伝える宣言を含む。複数のDLLが参照される場合には、複数のDLL定義サイドファイルが提供される。

【0066】アプリケーションがDLLをアクセスすることを可能にするサイドファイルの使用について述べてきたが、当業者には容易に明らかなように、これを達成するためには他の方法も存在する。例えば、プレリンカがDLLのコード領域から情報を直接読み出してもよい。

【0067】アプリケーション・ソース・ファイルのコンパイル・オペレーションにより生成されるオブジェクト・ファイルは、図1にAPP.objとして示されるように、DLLにより供給されるDLL定義サイドファイルをアクセスできなければならない。これを達成するために、DLLプロバイダはDLL定義サイドファイルに、プレリンク時にリンクするためのオブジェクト・デッキ

のセットを供給する。

【0068】DLLからエクスポートされた各関数または変数（コンパイラにより生成されるオブジェクト・デック内に示される）に対して、プレリンカは“IMPORT”プレリンカ宣言、すなわち、

```
IMPORT CODE <dll_name>; <identifier>;
```

または

```
IMPORT DATA <dll_name>; <identifier>;
```

を生成する。

【0069】このIMPORTサイドファイルはDLLプレリンクの間に生成され、DLLから記号をインポートする別のプログラムのプレリンクにおいて使用される。インポートするプログラムにとって、この宣言は記号をエクスポートするロード・モジュールの名前を提供する。

【0070】定義サイドファイルは、DLLのプレリンクの間にDLLコード自身の通常のオブジェクト・デックに加えて自動的に生成される。例えば、上記ソース・ファイル内のDLLに対して、プレリンカは次のDLL定義サイドファイルを生成する。

```
IMPORT CODE MATH math_func1
```

```
IMPORT CODE MATH math_func2
```

```
IMPORT DATA MATH math_var1
```

```
IMPORT DATA MATH math_var2
```

【0071】DLLプロバイダは、どの関数及び変数がインポートされるかを制御するDLL定義サイドファイルを編集することができる。例えば、上述例では、DLLプロバイダが“math_func2()”を露呈したくない場合、プロバイダはレコード“IMPORT CODE MATH math_func2”を定義サイドファイルから除去する。

【0072】アプリケーションのプレリンクの間、プレリンカは全てのインポート関数及び変数のリスト、並びにそれらがインポートされる対応するロード・モジュールを収集する。プレリンカはこれらのリストを定義するオブジェクト・コードを発行する。各リストは、関数または変数、記号名、アプリケーションのデータ領域内の記述子のオフセット、並びに記号がインポートされるDLLの名前を記述する情報を含む。

【0073】このリストは、アドレス取り決めを必要とする記述子を突き止めるために、DLLがロードされた後にRTLにより使用される。

【0074】本発明を実現するために、各DLLが特定の定義特性を有するように生成されなければならない。本発明ではこれらの特性をMVSオペレーティング・システムにおいて要求される形式で記述するが、他のオペレーティング・システム上における本発明の実現についても当業者には明らかであろう。

【0075】ロード時に、プログラムのコード領域及びデータ領域が生成される。RTLがシステム・ローダを用いてアプリケーションをロードし、コード領域内に静的に結合される記号のアドレスを解決する。アプリケー

ションがロードされる時、RTLがプレリンカにより収集及び生成されたコード領域内の情報を突き止め、それらを読み出し、そのプログラムに対応するデータ領域を割当て、初期化する。

【0076】この初期化データ領域の基底アドレスが、パラメータとしてアプリケーションに渡される。コンパイラにより生成されるコードが、この基底を用いて、アプリケーションのデータ領域内のオブジェクトのアドレスを計算する。

【0077】1度ロードされると、RTLは動的リンクを提供するために記述子内のアドレスを取り決める責任を負う。RTLは、ロードされたDLLのデータ領域内に存在するプレリンカにより生成された情報を通じ、エクスポート記号名及びアドレスを見い出す。

【0078】暗黙的ロード時DLLでは、DLLがロードされ、任意のユーザ・コードが実行される以前に、あらゆる対応する記述子がRTLにより取り決められる。これは“C”言語では、ファイル有効範囲内の次のような初期化子として発生する。

```
int * ip = &i; /*記号'i'のアドレスを'ip'に記憶*/
ここで'i'はインポートされる。'ip'は任意のユーザ・コードが実行される以前に'i'のアドレスを含まねばならない。これは'i'を定義するDLLが、暗黙的ロード時DLLでなければならないことを意味する。
```

【0079】暗黙的ロード・オン・ユースDLLは、ユーザ・コードの実行の間にロードされるDLLである。隠れRTLサービスが関数呼び出しによりトリガされる。RTLサービスは、記述子内に示されるDLLをロードし、制御をユーザ・コードに戻す以前に、あらゆる必要な記述子を取り決める。

【0080】ロード・オン・ユースDLL（最も要求的）は、標準のリンカ／ローダにより提供されない以下の4つの事項を要求する。

1. 要求時におけるDLLモジュールの自動的／暗黙的ロード。
2. DLLのロード時における、アプリケーション内の変数／関数参照とDLL内の対応定義との接続。
3. 別々のモジュール間での呼び出しに渡るデータ領域のスワップ（アプリケーションからDLLへ、またはDLLからDLLへ）。
4. ソースまたはモジュール準備（コンパイル、プレリンク、リンク）における、アプリケーション作成者またはDLLプロバイダからの最小の介入による上述の高速な実現。

【0081】本明細書で述べられる基本暗黙的ロード・オン・ユース機構は、これらの必要な事項を提供する。この機構には次の3つの主要要素が存在する。

【0082】1. （所与のアプリケーションにおける）変数及び関数記述子。所与のアプリケーションから参照され、特定のDLLまたはアプリケーション内で定義さ

れる各外部変数／関数に対して1つのこうした記述子が存在する。各記述子は、アプリケーションのデータ領域内の制御情報ブロックであり、所与の変数または関数に関する2つの事項を記述する。

a. 変数／関数がアプリケーション内で定義されていないときの、特定の変数／関数を所有するDLLのロード方法。

b. アプリケーションまたはDLL記憶内の特定の変数または関数のアドレス。そして関数の場合には、記述子は更に次の事項を記録する。

c. (DLLがロードされたときの) DLLのデータ領域のアドレス。

【0083】2. インポート変数を参照するコンパイラ生成DLLコード・シーケンス。コンパイラはインポート変数を参照するDLLコード・シーケンスを発行する。このシーケンスは最初に(変数がアプリケーション内に存在せず、DLL内に存在する場合) DLLがロードされることを保証し、DLLがロードされていなければDLLのロードをトリガする。次に、アプリケーションのデータ領域内の対応する変数記述子を介して間接的に所望の変数をアクセスする。

【0084】3. インポート関数の呼び出し。コンパイラは、関数がアプリケーション内の別のコンパイル単位内で定義されるか、または真にDLLからインポートされるかを知らず、両方において同一コードを生成する。呼び出される関数は関数記述子内に示される。任意のインポート関数に対して関数記述子は最初にRTLトリガ関数を示す。呼び出されると、トリガ関数はDLL(コード領域及びデータ領域の両方)をロードし、呼び出側の関数記述子内に関数アドレス及びデータ領域のアドレスを取り決める。それにより、続く呼び出しにおいては、最初に意図された関数が直接呼び出されることになる。DLLをロードし、関数記述子を変更した後、トリガ関数は最初に意図された関数を呼び出す。

【0085】これらの暗黙的ロード時並びにロード・オン・ユースDLLの主要特性は、それらの使用がアプリケーション・コードにとって実質的に透過的であることである。アプリケーションは単に、DLL関数の呼び出しを名前によりコード化するか、またはDLL変数の明示的な参照を名前によりコード化する。DLLの含意は、アプリケーション・ソース・コードのソース・レベル・ビュー(view)から多分に隠される。

【0086】参照とDLLからエクスポートされる定義との間の実際の接続は、DLLのロード時に形成される。DLLは、任意のユーザ・コードが実行される以前、または実行時におけるユーザ・コードの実行の間にロードされる。

【0087】本発明はロード時に関連して述べられ、これはIBMから提供されるMVSオペレーティング・システム上では、暗黙的ロード・オン・ユース・コマン

ド、または実行時DLLとして知られる。説明を容易にするため、無限定の用語“DLL”は一般に、暗黙的ロード時DLLまたは暗黙的ロード・オン・ユースDLLを意味するものとする。明示的実行時DLLの実現は、多分に暗黙的ロード時DLLの実現の拡張である。特に暗黙的ロード・オン・ユースDLLまたは明示的実行時DLLについて述べるときには、適宜参照を明示的に限定するか、文脈によりこれらを識別することにする。

【0088】明示的実行時DLLの1つの主要特性は、それらの使用がソース・レベルのアプリケーション・コードにより明示的に制御されることである。アプリケーションとDLL関数との間の接続は、1つ以上の実行時サービスの明示的なソース・レベル呼び出しを介して形成される。

【0089】明示的実行時DLLでは、プレリンカにより生成されるインポート及びエクスポート記号のリストがRTLにより使用され、要求関数または変数が見い出され、問合せアプリケーションに返却するための関数アドレスまたは変数アドレスが形成される。

【0090】暗黙的ロード時DLL及びロード・オン・ユースDLLでは、これらのエクスポート記号リストがDLLのロード時にRTLにより使用され、インポートするアプリケーションのデータ領域内のインポート変数記述子または関数記述子が初期化される。

【0091】実行時の実現は、比較的小さく良好に統合された拡張として、この基本暗黙的ロード・オン・ユース機構に追加される。

【0092】次に明示的実行時DLLサービスについて述べることにする。

【0093】明示的実行時サービスに関する本発明の使用例を次に示す。明示的実行時DLLを使用するためにソース・コード作成者は明示的に実行時サービスを呼び出す。実行時サービスの呼び出しを除けば、DLLモジュールへのリンクを要求するアプリケーション・モジュールと、アプリケーション・モジュールにリンクされるDLLとの間には、暗黙的または自動的接続は存在しない。例えば、アプリケーション作成者はソース・コード内に次の幾つかの呼び出しを含むことができる。

`dllload():DLLのハンドルを獲得`

`#include <dll.h>`

【0094】`dllhandle* dllload(char *dllName);` この関数は明示的アプリケーション要求に際し、DLLが以前にロードされていなければこれをメモリにロードし、呼び出したプログラムにそのDLLのハンドルを返却することにより、DLLをアプリケーションに接続する。このハンドルは、続く実行時DLL要求に際し、要求DLLを一意的に識別する。

【0095】

dllqueryfn(): D L L関数のアドレスを獲得

```
#include <dll.h>;  
void(*dllqueryfn(dllhandle * dh, char * funcName)) ();
```

【0096】この関数は、以前のdllload()要求から返却されたD L Lハンドル、及びそのD L L内のエクスポート関数の名前を受諾し、要求D L L関数を呼び出す実行可能スタブのアドレスを返却する。このスタブは実際の要求関数またはD L Lの一部であってもなくてもよ

く、どちらにも想定されない。返却アドレスだけが実行可能であり、最終的に要求関数を呼び出すことが保証される。dllqueryfn()の呼び出しが失敗すると、NULLアドレスが返却される。

【0097】

dllqueryvar(): D L L変数のアドレスを獲得

```
#include <dll.h>;  
void* dllqueryvar(dllhandle * dh, char * varName);
```

【0098】この関数は、以前のdllload()要求から返却されたD L Lハンドル、及びそのD L Lの特定のエクスポート変数の識別子を受諾し、指定D L Lの記憶内の要求変数のアドレスを返却する。dllqueryvar()の呼び出しが失敗すると、NULLアドレスが返却される。

【0099】アプリケーション作成者は、アプリケーション準備(すなわちコンパイルまたはプレリンク)の間にD L L定義サイドファイルを含む必要がない。しかしながら、アプリケーション作成者はD L L内の所望のエクスポート変数及び関数の名前の知識を有さねばならない。これは特定のD L Lプロバイダにより決定されるヘッダまたは他の手段(例えばプロダクト文書化)を介して獲得されうる。

【0100】アプリケーションはdllqueryvar()に掛けられる変数を、間接的に返却変数アドレス上でアクセスできる。また、返却関数アドレスを呼び出すことにより、dllqueryfn()に掛けられる関数を呼び出すことができる。

【0101】

dllfree(): D L Lのハンドルを解放

```
#include <dll.h>
```

【0102】この関数は提供されたD L Lハンドルを解放し、D L Lをメモリから適宜であれば(すなわち、これがD L Lをアクセスする最後のハンドルであれば)消去する。アプリケーションにより使用されるハンドルがdllfree()により解放された後に、dllqueryvar()に掛けられる変数またはdllqueryfn()に掛けられる関数が参照される場合、振舞いは定義されない。

【0103】D L Lへの実行時アクセスの場合の違いは、使用アプリケーションに完全に限定される。アプリケーションによるインポート関数及び変数の暗黙的参照が、R T Lサービスの明示的呼び出しdllload()、dllqueryfn() (またはdllqueryvar())により置換され、その後、dllqueryvar()またはdllqueryfn()呼び出しからの返却アドレスを通じ、これらのインポート関数及び変数が間接的に参照される。一方、D L Lプロバイダは上述のロード時の場合と全く同様にD L Lを準備する。

【0104】D L Lはエンクレープ(enclave)・レベ

ルで共有される。エンクレープはルーチンの独立な集まりである。ルーチンの1つがメインとして指定される。エンクレープは概略プログラムと類似である。これらのルーチンはアプリケーション内に存在したり、幾つかはアプリケーションがロードするD L L内に存在しうる。

【0105】参照されるD L Lは1エンクレープにつき正に1度ロードされ、1エンクレープ当たりの1 D L Lにつき、エクスポート変数の正に1コピーが生成/保持される。従って、D L Lの単一のコピーは、所与のエンクレープ内のモジュールを用いる全てにサービスする。これは所与のエンクレープ内でロード時及び実行時にアクセスされるD L Lにも当てはまる。特に、所与のエンクレープ内の所与のD L Lを、実行時D L L及びロード時D L Lとして(必要に応じて、複数の実行時ハンドルを介することにより)同時にアクセスすることが可能である。所与のエンクレープ内の所与のD L L内の所与の変数への全てのアクセスが、その変数の1コピーだけを参照する。同様に、所与のエンクレープ内の所与のD L L内の所与の関数への全てのアクセスが、その関数の1コピーだけを参照する。

【0106】1エンクレープにつきD L Lの1コピーだけが保持されるが、論理的ロードの回数はカウントされる。所与のエンクレープ内の所与のD L Lに対して、D L Lが暗黙的ロード・オン・ユースと指定される場合、1論理ロードがカウントされて、任意の参照がそのD L Lに対して実施され、またそのD L Lに対して発行されるあらゆるdllload()に対して1論理ロードがカウントされる。

【0107】D L Lの実際の消去(その変数を含む)は、その論理ロード・カウントが0に達するとき発生する。

【0108】D L Lは明示的実行時dllfree()要求により、またはエンクレープ終了時にロード済みロード時D L Lに対して発生する暗黙的解放により論理的に解放される。dllfree()は明示的実行時D L Lにとって任意選択的である。エンクレープ終了時に残るあらゆる(ロード時または実行時)D L L(すなわち、これらはロード時にアクセスされたために、または実行時にアクセスされたが十分なdllfree()が発行されなかったために、非

ゼロの使用カウントを有する)が、自動的に消去される(その変数を含む)。終了時におけるこの暗黙的解放は、ロード時DLLが解放されうる唯一の方法である。しかしながら、明示的実行時DLLは、明示的dllfree()により終了以前に解放されうる。それに対して実行時dllfree()要求を発行することにより、DLLのロード時コピーを(故意的または不注意に)解放することは不可能である。

【0109】RTLはロードされたDLLのリストを保持するので、必要に応じてそれらをロードし、また明示的呼び出しまたはアプリケーションの処分に際し、それらを解放することが可能である。

【0110】アプリケーションの終了に際し、RTLはあらゆる残りのDLLを消去する。

【0111】次に2つのエクスポートDLL及び1つのインポート・アプリケーションを構成する例について示すことにする。

【0112】図1において、アプリケーション・ソース・ファイルAPP.cが、ソース・ファイルMATH.c及びTRIG.cとして示される2つのDLLから関数及び変数を参照する。

【0113】ソース・ファイルMATH.cは、オブジェクト・デックMATH.objを生成するようにコンパイルされる。本発明を実現するために、オブジェクト・デックMATH.objがプレリンカによりプレリンクされる。

【0114】オブジェクト・デックMATH.objのプレリンクにより、オブジェクト・デックMATH.pobj及びDLL定義サイドファイルMATH.objdefが生成される。MATH.pobjはシステム・リンカによりリンクされ、実行可能ロード・モジュールMATH.loadを形成する。

【0115】ソース・ファイルTRIG.cも同様にコンパイルされ、プレリンクされて、オブジェクト・デックTRIG.pobj及びDLL定義サイドファイルTRIG.objdefを提供する。TRIG.pobjもまた、システム・リンカによりリンクされ、実行可能ロード・モジュールTRIG.loadを形成する。

【0116】ソース・ファイルAPP.cも同様にコンパイル及びプレリンクされて、オブジェクト・デックAPP.pobjを提供するが、DLL定義サイドファイルは生成されない。なぜならアプリケーションにより記号がエクスポートされないからである。DLL MATH及びTRIGに対応する2つのDLL定義サイドファイルがAPPのプレリンクに含まれ、記号のインポート先に関する情報を提供する。

【0117】上述の例では、各DLLが単一のコンパイル単位から構成されるように示される。しかしながら、DLLは複数のコンパイル単位から構成されてもよい。こうした状況では、各コンパイル単位が別々にコンパイルされる。しかしながら、単一のプレリンクだけが各DLLに対して要求される。また図1に示される例では、

アプリケーションAPPがそれ自体DLLではなく、MATH及びTRIGモジュールが他のDLLを使用しない。

【0118】アプリケーションがDLLを使用できるようにするため、アプリケーション作成者はアプリケーションに、DLLをアクセスするための適切な命令を提供しなければならない。本明細書では、再度、本発明がMVSオペレーティング・システムに適用されたときについて述べるが、当業者には、本発明が複数のソース・ファイルのリンクを可能にする他のシステムにおいて実現される方法についても、容易に明らかであろう。

【0119】図2はアプリケーションAPPとDLL MATH間の接続を示す。図2に示されるように、アプリケーションAPPは、システム・ローダがロードした書き込み保護される実行可能命令を有するコード領域を有する。DLL MATHも類似のコード領域及びデータ領域を有する。本発明では、RTLが従来システムの場合のように単にロード時だけではなく、プログラムの実行時にもデータ領域内の値を変更する。

【0120】インポート記号の記述子は、DLL MATHがアプリケーションAPPの実行中にロードされた後にRTLにより取り決められるアドレスを有する。MATHから記号をインポートする各プログラムは、その自身のデータ領域内に自身の記述子を有する。2つのプログラムが同一の記号をMATHからインポートする場合、それらのプログラムの各々は自身の記述子を有するが、MATH内のその記号に対する記述子内のアドレスは同一である。

【0121】また当業者には、本発明が他のDLLを使用するDLLをサポートすることも明らかであろう。

【0122】まとめとして、本発明の構成に関して以下の事項を開示する。

【0123】(1) 動的にリンクされるライブラリをデータ処理オペレーティング・システム上で実行されるプログラムにリンクする改良された方法であって、前記データ処理オペレーティング・システムがコンパイラ、プレリンカ、実行時ライブラリ、システム・リンカ及びシステム・ローダを有するものにおいて、各動的リンク・ライブラリに対応して、前記プレリンカにより、前記ライブラリ内で使用可能な外部記号のセットに関する情報を含む定義ファイルを生成するステップであって、前記プレリンカが前記定義ファイルを前記動的リンク・ライブラリ内の任意の変更を反映して最新に維持する、前記生成ステップと、前記コンパイラによりアプリケーション・プログラム・コードをコンパイルするステップであって、前記コンパイル・コードがトリガ・ルーチン呼び出すコードを含む、前記コンパイル・ステップと、前記アプリケーション・プログラム・コードを前記定義ファイルにプレリンクするステップであって、エクスポート定義セクションを含み、前記システム・リンカによりリンクされるオブジェクト・デックを生成する、前記プレリンク・ステップと、前記システム・リンカにより、

前記アプリケーション・プログラム・コードをリンクするステップと、前記コードを実行し、前記トリガ・ルーチンにより前記コード内の未解決記号の参照を解決するステップであって、前記トリガ・ルーチンが前記プレリンカにより生成された前記エクスポート定義セクションをアクセスすることにより、前記実行時ライブラリが前記未解決記号を含む前記動的リンク・ライブラリをアクセスし、前記動的リンク・ライブラリをロードし、前記記号参照を解決することを可能にし、前記システム・リンクまたはローダの支援を必要とせずに、前記アプリケーション・プログラムに前記記号を提供する、前記実行及び解決ステップと、を含む、方法。

(2) 前記外部記号が外部変数及び外部関数を含む、前記(1)の方法。

(3) 前記トリガ・ルーチンが変数の参照に関するトリガ・ロードまたは関数の呼び出しに関するトリガ・ロードである、前記(1)の方法。

(4) 前記定義ファイルが各特定の記号がコードかデータかを指定し、前記記号を定義する前記動的リンク・ライブラリを指定する、前記(1)、(2)または(3)の方法。

(5) 前記エクスポート定義セクションが前記定義ファイルからの情報を含む、前記(1)、(2)または(3)の方法。

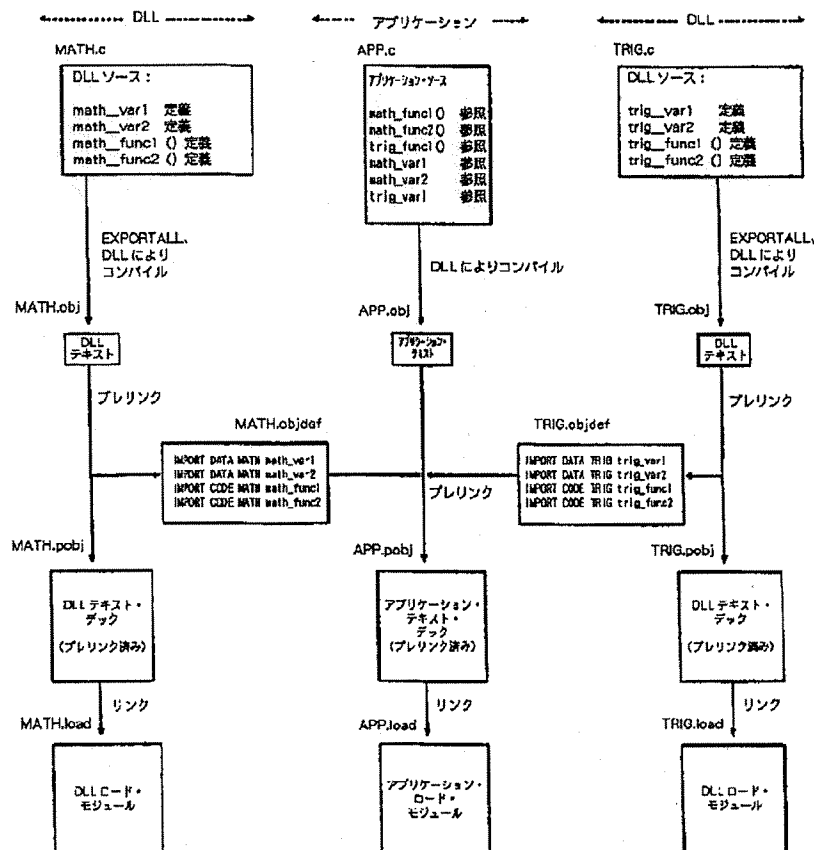
【図面の簡単な説明】

【図1】本発明の教示により、2つのDLLモジュール及び関連アプリケーション・モジュールが生成される様子を示す図である。

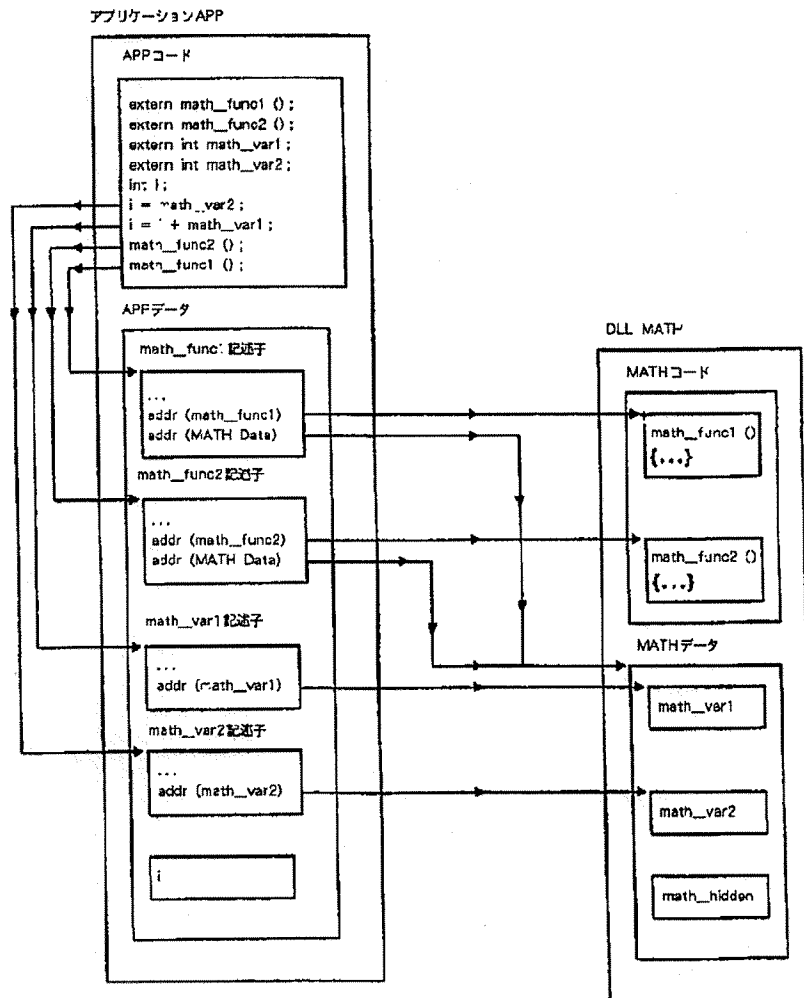
【図2】本発明の使用時における関数及び変数のインポートとエクスポート間の関係を示す図である。

【図3】本発明が以前に作成されたコードの再コンパイルを必要とせずに、旧コードの使用を可能にする様子を示す図である。

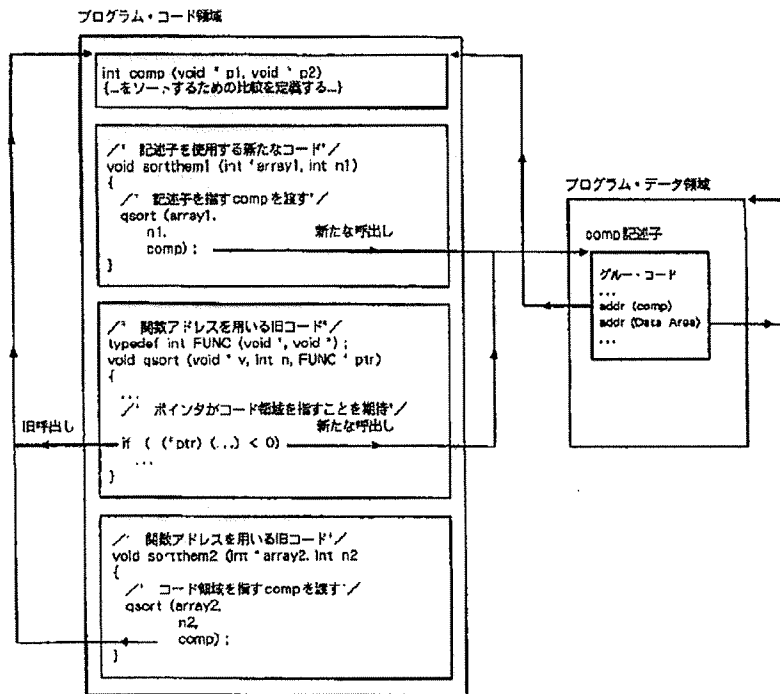
【図1】



【図2】



【図 3】



フロントページの続き

(72)発明者 マイク・ステファン・フルトン
カナダ、オンタリオ州ドン・ミルズ、ユニ
ット87、バレイウッズ・ロード 43

(72)発明者 ブライアン・ジェームズ・ミンチャウ
カナダ、オンタリオ州ノース・ヨーク、ワ
ゴン・トレイルウェイ 11